Divide-and-conquer generic recurrence and the "Master Theorem"

Vassos Hadzilacos

Consider a typical divide-and-conquer algorithm that, at each stage, divides up the input of size n into a smaller inputs of size n/b each, recursively finds the answer for each smaller input, and combines the answers into the answer for the original input. The running time of such an algorithm can be expressed by the following recurrence, where we assume for simplicity that n is a power of b:

$$T(n) = \begin{cases} a \cdot T(n/b) + c \cdot n^d, & \text{if } n > 1 \\ c, & \text{if } n = 1, \end{cases} \tag{1}$$

where $a \geq 1$ and $b > 1$ are integer constants and $c, d \geq 0$ are real constants (i.e., a, b, c and d do not depend on n). The first term, $a \cdot T(n/b)$, represents the time required for the a recursive calls, each to an input of size n/b; and the second term, $c \cdot n^d$, represents the time required to divide up the input into a pieces of size n/b each and to combine the results of the recursive calls into the result of the main call. The power d depends on the algorithm at hand. For example, if dividing the input and combining the answer takes constant time, $d = 0$; if it takes linear time, then $d = 1$; and so on.

The so-called "Master Theorem" gives the solution to the above recurrence in a convenient form, and is a very useful tool that simplifies tremendously the running time analysis of many divide-and-conquer algorithms.

Theorem 1 *If $T(n)$ is the function defined by recurrence (1), where $a \geq 1$ and $b > 1$ are integer constants, $c, d \geq 0$ are real constants, and n is a power of b, then*

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \log n), & \text{if } a = b^d \\ \Theta(n^{\log_b a}), & \text{if } a > b^d. \end{cases} \tag{2}$$

(Note that the constant c in the definition of $T(n)$ does not appear explicitly in (2): It is "absorbed" by the constants implicit in the Θ-notation.)

Proof. This can be easily checked by using induction. A more insightful proof, however, can be obtained by reasoning about the amount of time required by the divide and conquer algorithm whose running time is represented by (1).

Consider the tree of recursive calls made by the algorithm:

- At level 0 (the root), we have one call for input of size n. The time needed by this call, exclusive of the time needed by the calls it makes (i.e., the time to divide up its input and to combine the results of the calls it makes), is cn^d — the second term in the definition of the recurrence.
- At level 1 we have the calls made by the call at level 0: there are a such calls (one for each subproblem), each working on an input of size n/b. The time needed by *each* of these calls, exclusive of the time needed by the calls it makes, is $c(n/b)^d$, so the total time needed by these a calls is $ac(n/b)^d$.

- At level 2 we have the calls made by the calls at level 1: there are a^2 such calls, each working on an input of size n/b^2. Reasoning as before, the total time needed by these a^2 calls is $a^2c(n/b^2)^d$.
- In general, at level i, we have a^i calls, each working on an input of size n/b^i. The total time needed for these calls is $a^ic(n/b^i)^d$.
- This continues for every integer $i = 0, 1, 2, \ldots, \log_b n$. For $i = \log_b n$, the input size is 1, and we have reached the base case of the recursion.

Thus, the total time required for all the calls at all levels is

$$T(n) = \sum_{i=0}^{\log_b n} a^i c\Big(\frac{n}{b^i}\Big)^d = cn^d \sum_{i=0}^{\log_b n} \Big(\frac{a}{b^d}\Big)^i. \tag{3}$$

Note that $\sum_{i=0}^{\log_b n}(a/b^d)^i$ is just a geometric series of the form $\sum_{i=0}^{k} x^i$. From basic mathematics recall that

- if $0 \le x < 1$, then $\sum_{i=0}^{\infty} x^i = 1/(1-x)$, and
- for all $x \ge 0$, $\sum_{i=0}^{k} x^i = (x^{k+1}-1)/(x-1)$.

So we have:

Case 1. $a < b^d$. Then, by appealing to the formula for the infinite geometric series, we have:

$$\sum_{i=0}^{\log_b n} \Big(\frac{a}{b^d}\Big)^i < \sum_{i=0}^{\infty} \Big(\frac{a}{b^d}\Big)^i = 1/(1-a/b^d) = \Theta(1).$$

So by (3), $T(n) = \Theta(n^d)$ in this case.

Case 2. $a = b^d$. Then we have:

$$\sum_{i=0}^{\log_b n} \Big(\frac{a}{b^d}\Big)^i = \sum_{i=0}^{\log_b n} 1^i = 1 + \log_b n = \Theta(\log n).$$

So by (3), $T(n) = \Theta(n^d \log n)$ in this case.

Case 3. $a > b^d$. Then, by appealing to the formula for the finite geometric series, we have:

$$\sum_{i=0}^{\log_b n} \Big(\frac{a}{b^d}\Big)^i = \frac{\Big(\frac{a}{b^d}\Big)^{1+\log_b n} - 1}{\underbrace{\frac{a}{b^d} - 1}_{\Theta(1)}} = \Theta\left(\frac{a^{\log_b n}}{\big(b^{\log_b n}\big)^d}\right) = \Theta\left(\frac{n^{\log_b a}}{n^d}\right).$$

(In the last step we use the following facts about logarithms: $a^{\log_b n} = n^{\log_b a}$ (verify by taking logarithms of both sides), and $b^{\log_b n} = n$ (by definition of logarithm).) So by (3), $T(n) = \Theta(n^{\log_b a})$ in this case. □

Arbitrary values of n

In the preceding analysis we assumed that the input size n is a positive integer that is a power of $b > 1$, and therefore that b is also an integer. It turns out that, essentially, the Master Theorem holds even if n is not necessarily a power of b and $b > 1$ is a real number, not necessarily an integer. If n is not a power of b, however, (1) is not a legitimate recurrence: sooner or later as we keep dividing the input size by b we will end up with a non-natural number and then the recurrence is not defined. Also, a non-natural number as an input size does not make sense.

In general, a divide-and-conquer algorithm breaks a problem of size n into a subproblems, a_1 of which have size $\lceil n/b \rceil$ and a_2 have size $\lfloor n/b \rfloor$, for some non-negative integers a_1 and a_2 such that $a_1 + a_2 = a$. Thus, the following recurrence describes the running time of such an algorithm, when n is not necessarily a power of b, and $b > 1$ is not necessarily an integer:

$$T(n) = \begin{cases} a_1 \cdot T(\lceil n/b \rceil) + a_2 \cdot T(\lfloor n/b \rfloor) + c \cdot n^d, & \text{if } n \geq b/(b-1) \\ c, & \text{if } 1 \leq n < b/(b-1), \end{cases} \tag{4}$$

where a_1, a_2 are non-negative integers such that $a_1 + a_2 = a$. (The requirement that $n \geq b/(b-1)$ in the first case of the recursive definition ensures that $\lceil n/b \rceil < n$ and therefore that the recurrence is well-founded.) It turns out that this recurrence also satisfies (2). This can be verified by induction, though the algebra is somewhat daunting. The proof below uses an alternative approach.

Theorem 2 (Master Theorem) *If $T(n)$ is the function defined by recurrence (4), where a_1, a_2 are non-negative integers such that $a_1 + a_2 = a \geq 1$, and $b > 1, c \geq 0, d \geq 0$ are reals, then $T(n)$ satisfies (2).*

Proof. Let $T' : \mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$ be the function (over the non-negative reals) defined by:

$$T'(n) = \begin{cases} a_1 \cdot T'(\lceil n/b \rceil) + a_2 \cdot T'(\lfloor n/b \rfloor) + c \cdot n^d, & \text{if } n \geq b/(b-1) \\ c, & \text{if } n < b/(b-1), \end{cases}$$

First we have to argue that this does, in fact, define a function: recursion can be used to define a function over the natural numbers, but now our domain is the reals, and we cannot appeal directly to induction to justify that the function is properly defined. Note, however, that if n is an integer then $T'(n) = T(n)$ and since T is well-defined, T' is well defined when restricted to integers. Furthermore, for any real number n, whether integer or not, $T'(n)$ is determined uniquely by the values of function T' at integer arguments, which as we just mentioned are well-defined. Therefore T' is a well-defined function for all reals.

Using a straightforward induction it is easy to check that T (the recursively defined function over the integers) is non-decreasing. From this and the fact that the ceiling, floor, and cn^d functions are non-decreasing, it follows that T' is also non-decreasing.

Having established that T' is a well-defined non-decreasing function over the non-negative reals, we now define the function $S : \mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$ as

$$S(n) = T'(n + \kappa)$$

for some real constant κ. For reasons that will become clear shortly, we will choose $\kappa = b/(b-1)$; for now we treat it as a constant "to-be-determined".

First we will obtain an upper bound on S. We have:

$$\begin{aligned}
S(n) &= T'(n+\kappa) && \text{[by the definition of } S\text{]}\\
&= a_1 T'\big(\lceil (n+\kappa)/b \rceil\big) + a_2 T'\big(\lfloor (n+\kappa)/b \rfloor\big) + c(n+\kappa)^d && \text{[by the definition of } T'\text{]}\\
&\le aT'\big((n+\kappa)/b + 1\big) + c(n+\kappa)^d && \text{[since } T' \text{ is non-decreasing, and}\\
& && \text{for any real } x,\ \lceil x \rceil < x+1\text{]}\\
&\le aT'(n/b + \kappa/b + 1) + c'n^d && \text{[for some constant } c', \text{ since } \kappa \text{ is a constant]}\\
&= aT'(n/b + \kappa) + c'n^d && \text{[by choosing } \kappa = b/(b-1)\text{]}\\
&= aS(n/b) + c'n^d && \text{[by definition of } S\text{]}
\end{aligned}$$

Summarizing the above inequalities we have

$$S(n) \le aS(n/b) + c'n^d.$$

Applying this inequality repeatedly, and using the geometric series formulas as in the proof of Theorem 1 we get that

$$S(n) = \begin{cases} O(n^d) & \text{if } a < b^d \\ O(n^d \log n), & \text{if } a = b^d \\ O(n^{\log_b a}), & \text{if } a > b^d. \end{cases}$$

By definition of S, $T'(n) = S(n-\kappa)$. Therefore (keeping in mind that $\kappa = b/(b-1)$ is a constant),

$$T'(n) = \begin{cases} O\big((n-\kappa)^d\big) = O(n^d) & \text{if } a < b^d \\ O\big((n-\kappa)^d \log(n-\kappa)\big) = O(n^d \log n), & \text{if } a = b^d \\ O\big((n-\kappa)^{\log_b a}\big) = O(n^{\log_b a}), & \text{if } a > b^d. \end{cases} \tag{5}$$

We can also obtain a lower bound on S, and thereby on T', by applying similar reasoning but using floors instead of ceilings in the definition of T':

$$T'(n) = \begin{cases} \Omega(n^d) & \text{if } a < b^d \\ \Omega(n^d \log n), & \text{if } a = b^d \\ \Omega(n^{\log_b a}), & \text{if } a > b^d. \end{cases} \tag{6}$$

Theorem 2 now follows from (5) and (6), since T coincides with T' on integer values. □